Strategy Synthesis for Autonomous Agents Using PRISM

Ruben Giaquinta[1], Ruth Hoffmann[3], Murray Ireland[2], Alice Miller[1], and Gethin Norman[1(✉)]

[1] School of Computing Science, University of Glasgow, Glasgow, UK
gethin.norman@glasgow.ac.uk
[2] School of Engineering, University of Glasgow, Glasgow, UK
[3] School of Computer Science, University of St Andrews, St Andrews, UK

Abstract. We present probabilistic models for autonomous agent search and retrieve missions derived from Simulink models for an Unmanned Aerial Vehicle (UAV) and show how probabilistic model checking and the probabilistic model checker PRISM can be used for optimal controller generation. We introduce a sequence of scenarios relevant to UAVs and other autonomous agents such as underwater and ground vehicles. For each scenario we demonstrate how it can be modelled using the PRISM language, give model checking statistics and present the synthesised optimal controllers. We conclude with a discussion of the limitations when using probabilistic model checking and PRISM in this context and what steps can be taken to overcome them. In addition, we consider how the controllers can be returned to the UAV and adapted for use on larger search areas.

1 Introduction

Autonomous vehicles such as unmanned aerial vehicles, autonomous underwater vehicles and autonomous ground vehicles have widespread application in both military and commercial contexts. Investment in autonomous systems is growing rapidly, the UK government is investing £100 million into getting driverless cars on the road, while the worldwide market for commercial applications of drone technology has been valued at over $127 billion. For example, the U.S. Office of Naval Research has demonstrated how a swarm of unmanned boats can help to patrol harbours, the Defence Advanced Research Projects Agency has launched a trial of the world's largest autonomous ship and NASA has deployed Mars Rovers which, on receipt of instructions to travel to a specific location, must decide on a safe route.

Understandably, there are concerns about safety and reliability of autonomous vehicles. Recently researchers exposed design flaws in drones by deliberately hacking their software and causing them to crash [34], and US regulators discovered that a driver was killed while using the autopilot feature of a Tesla car due to the failure of the sensor system to detect another vehicle.

A. Dutle et al. (Eds.): NFM 2018, LNCS 10811, pp. 220–236, 2018.
https://doi.org/10.1007/978-3-319-77935-5_16

Incidents like these and the lack of design and analysis tools to prove system compliance under all nominal and adverse operating conditions are preventing regulatory bodies from issuing clear certification guidelines.

Autonomous agents almost always follow a variation of the same core process: *perception*, *cognition* and *actuation*. Perception is achieved through the system sensor suite, giving the agent a picture of the current environmental state. Actuation governs how the agent interacts with the environment and cognition is where the agent decides at run-time what goals to set and how to achieve them. A critical question is how to implement this decision-making process. Current best-practice uses a software *controller* that pre-determines the behaviour of the agent under a given set of internal parameter values and environmental conditions. However, *can controllers be generated automatically and in such a way as to ensure that the resulting behaviour is safe, efficient and secure under all conceivable operational scenarios and system failures?*

Guaranteeing reliability of autonomous controllers using testing alone is infeasible, e.g. [15] concludes that autonomous vehicles would need to be driven hundreds of billions of miles to demonstrate their reliability and calls for the development of innovative methods for the demonstration of safety and reliability that could reduce the infeasible burden of testing. Formal verification offers hope in this direction having been used both for controller synthesis and for verifying the reliability and safety of autonomous controller logic. In this paper we investigate the use of probabilistic model checking and the probabilistic model checker PRISM for automatic controller generation. Our ultimate goal is to develop software, based on the techniques described here that can be embedded into controller software to generate *adaptable* controllers that are *verified* to be *optimal*, *safe* and *reliable* by design. Specifically we:

1. describe PRISM models for a suite of scenarios inspired by situations faced by a range of autonomous agents;
2. present synthesised (optimal) controllers for the different scenarios and examine their performance;
3. discuss the limitations of this approach and the next steps to overcome them.

Related Work. There has been significant recent work using Markov decision processes, temporal logic specifications and model checking for generating controllers of autonomous systems. These works differ in the temporal logic specifications used and include approaches using the branching time logic PCTL [22,36], linear time temporal logic LTL [7,35], metric temporal logic [11], rewards [30] and multi-objective queries [21,23]. Also, both partially observable Markov decision processes, e.g. [29,31], and stochastic games, e.g. [8,32] have been used in conjunction with temporal logic for controller synthesis of autonomous agents.

Formal verification of robot missions is considered in [27], however here the focus is on evaluating existing controllers. Similarly, in [6] model checking is used to verify the decision making aspect of autonomous systems. Model checking has also been used for analysing the behaviour of groups or swarms of autonomous agents including: agents in a pursuer-evader [2] scenario, foraging swarms [18,24],

co-operative missions [13] and surveillance and convoy missions [5]. Concerning using formal verification to obtain certification of correctness, [33] uses (non-probabilistic) model checking to verify unmanned aircraft system controllers against the Civil Aviation Authority's regulations.

In previous work [12] we have presented a PRISM model of a UAV with a fixed controller searching for objects in a defined, gridded area following a fixed path. The real parameters are derived from a simulation model and the process of property verification using PRISM is compared to Monte Carlo simulation. This model is described in Scenario 1 (see Sect. 3).

2 Background

We now introduce Markov decision processes (MDPs) and probabilistic model checking of MDPs in PRISM. For any finite set X, let $Dist(X)$ denote the set of discrete probability distributions X.

Markov Decision Processes. MDPs model discrete time systems that exhibit both nondeterministic and probabilistic behaviour.

Definition 1. *A* Markov decision process *(MDP) is a tuple* $\mathsf{M} = (S, \bar{s}, A, P)$ *where: S is a finite set of states and $\bar{s} \in S$ is an initial state; A is a finite set of* actions*; $P : S \times A \to Dist(S)$ is a (partial) probabilistic transition function, mapping state-action pairs to probability distributions over S.*

In a state s of an MDP M, there is a nondeterministic choice between the *available* actions in s. These available actions, denoted $A(s)$, are the actions for which $P(s, a)$ is defined. If action a is selected, then the successor state is chosen probabilistically, where the probability of moving to state s' equals $P(s, a)(s')$. An execution of an MDP is a path corresponding to a sequence of transitions of the form $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$, where $a_i \in A(s_i)$ and $P(s_i, a_i)(s_{i+1}) > 0$ for all $i \geq 0$. Let $FPaths_{\mathsf{M}}$ denote the finite paths of M and $last(\pi)$ denote the last state of any finite path π.

Reward structures model quantitative measures of an MDP which are accumulated when an action is chosen in a state.

Definition 2. *A* reward structure *for an MDP* $\mathsf{M} = (S, \bar{s}, A, P)$ *is a function of the form $R : S \times A \to \mathbb{R}_{\geq 0}$.*

To reason about the behaviour of an MDP, we need to introduce the definition of *strategies* (also called policies, adversaries and schedulers). A strategy resolves the nondeterminism in an MDP by selecting the action to perform at any stage of execution. The choice can depend on the history and can be made randomly.

Definition 3. *A* strategy *of an MDP* M *is a function $\sigma : FPaths_{\mathsf{M}} \to Dist(A)$ such that $\sigma(\pi)(a) > 0$ only if $a \in A(last(\pi))$.*

Under a strategy σ of an MDP M, the nondeterminism of M is resolved, and hence its behaviour is fully probabilistic. Formally, it corresponds to an (infinite) state discrete time Markov chain (DTMC) and we can use a standard construction on DTMCs [17] to build a probability measure over the infinite paths of M.

Property Specifications. Two standard classes of properties for MDPs are *probabilistic* and *expected reachability*. For a given state predicate, these correspond to the probability of eventually reaching a state satisfying the predicate and the expected reward accumulated before doing so. The value of these properties depends on the resolution of the nondeterminism, i.e. the strategy, and we therefore consider optimal (minimum and maximum) values over all strategies.

The Probabilistic Model Checker PRISM. PRISM [19] is a probabilistic model checker that allows for the analysis of a number of probabilistic models including MDPs. Models in PRISM are expressed using a high level modelling language based on the Reactive Modules formalism [1]. A model consists of a number of interacting modules. Each module consists of a number of finite-valued variables corresponding to the module's state and the transitions of a module are defined by a number of guarded commands of the form:

$$[\texttt{<action>}]\ \texttt{<guard>}\ \rightarrow\ \texttt{<prob>} : \texttt{<update>} + \cdots + \texttt{<prob>} : \texttt{<update>}$$

A command consists of an (optional) action label, guard and probabilistic choice between updates. A guard is a predicate over variables, while an update specifies, using primed variables, how the variables of the module are updated when the command is taken. Interaction between modules is through guards (as guards can refer to variables of all modules) and action labels which allow modules to synchronise. Support for rewards are through reward items of the form:

$$[\texttt{<action>}]\ \texttt{<guard>} : \texttt{<reward>};$$

representing the reward accumulated when taking an action in a state satisfying the guard.

PRISM supports the computation of an optimal probabilistic and expected reachability values, for details on how these values are computed and the temporal logic that PRISM supports, see [10]. PRISM can also synthesise strategies achieving such optimal values. Such a strategy is represented as a list of (optimal) action choices for each state of the MDP under study, this list can then be fed back into PRISM to generate the underlying DTMC, and hence allow further analysis of the strategy. For details on strategy synthesis see [20].

3 Scenarios

We describe a number of scenarios relevant for autonomous agents and how PRISM has been used for verification and controller synthesis. Each scenario is inspired by realistic situations for a range of autonomous vehicle applications, e.g. border patrol using autonomous vehicles, exploration of unexplored terrain, and search and rescue operations. However, in each case we present a simplified scenario involving an *autonomous agent* involving search within a defined area. The PRISM model and property files for each scenario are available from [37].

Scenario 1: Fixed Controller. In [12] we introduced abstract PRISM models representing an agent searching for and collecting objects randomly placed in a grid. The models are based on a physical system, namely a quadrotor UAV in operation inside a small, constrained environment in the University of Glasgow's Micro Air Systems (MAST) Laboratory, a cuboidal flight space with a motion capture system for tracking UAVs. Continuous Simulink simulation models have been developed so that the effect of altering various aspects (including the search strategy) can be investigated via Monte Carlo simulation [14]. The purpose was to investigate the viability of a framework for analysing autonomous systems using probabilistic model checking of an abstract model where quantitative data for abstract actions is derived from small-scale simulation models.

The controller in this scenario is fixed and specifies that the agent searches the grid in a predetermined fashion, starting at the bottom left cell of the grid, travelling right along the bottom row to the bottom right cell, then left along the second row, and so on. The controller also specifies that if an object is found during search, then the agent attempts to pick up the object and, if successful, transports it to a specified deposit site. Whenever the agent's battery level falls below a specified threshold, it returns to the base to recharge and once the battery is charged resumes the search. In both cases, search resumes from the previous cell visited, until all objects have been found or because the search can not continue (e.g. due to an actuator fault or the mission time limit has been reached).

We used MDP models and PRISM to analyse this scenario with a grid size of 7×4 and either 2 or 3 objects. Although the controller is fixed, nondeterminism is used to represent uncertainty in the environment, specifically the time taken for the agent to execute actions, which were obtained from our small-scale simulation models. The PRISM models contain modules for the agent's behaviour, movement, time and battery level, and objects. To reduce the size of the state-space, rather than encoding the random placement of the objects within the model, we develop a model were objects have fixed coordinates and consider each possible placement of the objects. For example, in the case of two objects there are 378 different possible placements for the objects and each model with fixed placement has approximately 200,000 states. To obtain quantitative verification results for the model where objects are randomly placed we perform multiple verification runs by considering each possible placement of the objects and take an average.

In the remainder of the section we synthesise optimal *controllers* for different scenarios with respect to the mission time. We achieve this using PRISM to encode the choices of the controller using nondeterminism. We remove the nondeterminism corresponding to environmental factors, e.g. the time taken to perform actions as these are not choices of the controller. By moving to stochastic games [28] we could separate the controller's choices from that of the environment. However, implementations of probabilistic model checking for such games, e.g. PRISM-games [4], do not currently scale to the size of models we consider.

Scenario 2: Control of Recharging. In this scenario we introduce choice as to when the battery is recharged. More precisely, recharging is no longer enforced when the battery reaches a pre-determined lower threshold as in Scenario 1, but can be performed nondeterministically at any time during search. We assume that positions of the objects are fixed and the agent explores the grid in the pre-determined fashion described for Scenario 1 above.

Table 1. Scenario 2: performance of optimal and Scenario 1 controllers (7×4 grid).

Base	Depot	Object 1	Object 2	Expected mission time		Expected no. of battery charges		Probability of mission success	
				Scenario 1	Optimal	Scenario 1	Optimal	Scenario 1	Optimal
(0,0)	(2,2)	(3,3)	(4,3)	285.8	138.0	2.301	2.021	0.949	0.975
(0,0)	(2,2)	(1,1)	(4,3)	252.2	147.3	2.181	1.002	0.956	0.975
(0,0)	(6,3)	(2,1)	(4,2)	292.4	178.5	2.364	1.056	0.948	0.970
(1,2)	(6,3)	(3,0)	(5,1)	212.4	83.93	1.015	0.203	0.962	0.987
(3,2)	(6,3)	(2,1)	(4,2)	218.9	117.3	1.127	1.032	0.960	0.980
(6,3)	(0,0)	(2,1)	(4,2)	221.4	119.5	1.090	1.021	0.960	0.978

We use PRISM to find the minimum expected mission time and synthesise an optimal strategy that achieves this minimum for a suite of models involving two objects, varying the positions of the base, depot and objects. The synthesised strategies demonstrate that the optimal choice is to recharge when close to base, rather than waiting for the battery level to reach a threshold level. The performance of the synthesised optimal controller is compared to that of the fixed controller used in Scenario 1 (which recharges when the battery level reaches a threshold) in Table 1. The results demonstrate that the synthesised controller offers a significant performance improvement over the controller of Scenario 1. We see the expected mission time drastically reduces, the probability of a successful mission increases and the expected number of battery recharges decreases.

Scenario 3: Control of Search. We now generalise Scenario 2 to include control of the search path as well as recharging. Since allowing freedom of movement increases the complexity of our model, we focus on the search mode of the agent and abstract other modes (including take-off, hover and grab, see [14] for details).

Having the positions of the objects as constants in the PRISM model is not feasible if our aim is to generate optimal and realistic controllers as this means that the agent knows the locations of the objects it is searching for. In such a situation, the optimal search strategy is clear: go directly to the objects and collect them. We initially considered using partially observable MDPs (POMDPs) and the recent extension of PRISM [26]. Using POMDPs we can hide the positions of the objects and synthesise an optimal controller, e.g. one that minimises the

```
// number of unexplored cells
formula n = gp0+gp1+gp2+gp3+gp4+gp5+gp6+gp7+gp8+gp9+gp10+gp11;
// probability of finding object in an unexplored cell
formula p = objs/n;
```

Fig. 1. PRISM code: probability of finding an object in an unexplored cell.

```
// move east
[east] s=0 & gp=0 & gp0=0 & posx<X → (posx'=posx+1);
[east] s=0 & gp=0 & gp0=1 & posx<X & p≤1 → p : (s'=1)&(posx'=posx+1)&(gp0'=0)
                                          + 1−p : (posx'=posx+1)&(gp0'=0);
// move west
[west] s=0 & gp=0 & gp0=0 & posx>0 → (posx'=posx−1);
[west] s=0 & gp=0 & gp0=1 & posx>0 & p≤1 → p : (s'=1)&(posx'=posx−1)&(gp0'=0)
                                          + 1−p : (posx'=posx−1)&(gp0'=0);
// move north
[north] s=0 & gp=0 & gp0=0 & posy<Y → (posy'=posy+1);
[north] s=0 & gp=0 & gp0=1 & posy<Y & p≤1 → p : (s'=1)&(posy'=posy+1)&(gp0'=0)
                                          + 1−p : (posy'=posy+1)&(gp0'=0);
// move south
[south] s=0 & gp=0 & gp0=0 & posy>0 → (posy'=posy−1);
[south] s=0 & gp=0 & gp0=1 & posy>0 & p≤1 → p : (s'=1)&(posy'=posy−1)&(gp0'=0)
                                          + 1−p : (posy'=posy−1)&(gp0'=0);
```

Fig. 2. PRISM commands: searching cell with gridpoint 0.

expected mission time. However, we found the prototype implementation did not scale as it implements only basic analysis techniques.

Subsequently we investigated modelling hidden objects with MDPs. This was found to be feasible by monitoring the unexplored cells and using the fact that the probability of an object being found in a cell that has not been explored is obj/n were obj is the number of objects still to be found and n the number of unexplored cells. In an $M \times N$ grid we associate the cell with coordinates (x, y) the integer (or *gridpoint*) $x + y \cdot M$. Before we give the PRISM code for an agent searching, we list the variables, formulae and constants used in the code:

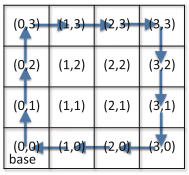
- variable s is the state of the agent taking value 0 when searching and 1 when an object has been found;
- variables $posx$ and $posy$ are the current coordinates of the agent and formula gp returns the corresponding gridpoint;
- constants X and Y represent the grid size, where $X = M-1$ and $Y = N-1$;
- variable gpi for $0 \leq i \leq (X+1) \times (Y+1) - 1$ equals 1 when cell with gridpoint i has not been visited, and 0 otherwise;
- variable $objs$ represents the number of objects yet to be found.

We assume the base and depot are fixed and located at position $(0, 0)$.

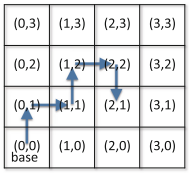
Figures 1 and 2 give the PRISM code extracts relevant for finding an object for a grid with 12 cells when the agent is searching the cell with gridpoint 0. To search the cell the agent needs to be searching and located in the cell ($s = 0$ and $gp = 0$). If the cell has already been searched ($gp0 = 0$), then there is just a nondeterministic choice as to which direction to move. If the cell has not been

Table 2. Scenario 3: model checking results.

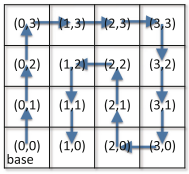
Grid size	No. of objects	Battery capacity	States	Transitions	Min expected mission time	Verification time (s)
4×4	1	24	403,298	1,016,387	24.00	1.248
4×4	2	24	860,689	2,212,391	38.60	2.840
5×4	1	28	5,332,892	13,942,821	29.20	14.94
5×4	2	28	11,841,031	31,526,709	46.27	38.77
6×4	1	32	64,541,199	172,990,992	34.33	197.4
6×4	2	32	149,723,921	408,008,297	53.94	514.5



(a) path 1 and recharge



(b) path 2 after recharge



(c) path (no recharge)

Fig. 3. Scenario 3: optimal controllers for 4×4 grid, battery capacities 24 and 28.

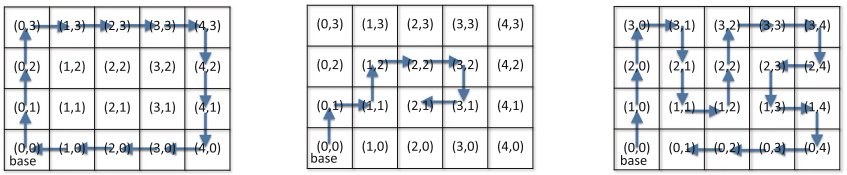
searched ($gp0 = 1$), then each choice includes the probability of finding an object using the formula in Fig. 1. The guard $p \leq 1$ prevents PRISM reporting modelling errors due to potentially negative probabilities. Boundaries are encoded in guards rather than using knowledge of the grid, e.g. it is not possible to move south or west in gridpoint 0, to allow automated model generation for different grids.

After an object has been found ($s = 1$), the agent deposits it at the base and resumes search if there are more objects to find. Returning to base either to deposit or recharge is encoded by a single transition with time and battery consumption updated assuming the controller takes a shortest path to the base. This modelling choice is to reduce the state space. Also to reduce the state space, we add conditions to guards in the battery module to prevent the agent moving to a position from which it cannot get to base with the remaining battery power. For example, for a 5×4 grid, two objects and a battery capacity of 28, together these modelling choices reduce the state space from 24,323,956 to 11,841,031.

We synthesised optimal strategies for the minimum expected mission time for grids of varying sizes and number of objects. Table 2 presents model checking results in which we have chosen the minimum battery size that allows for a successful mission for the given grid. Figures 3(a)–(b) and 4(a)–(b) present the optimal strategies when searching for a single object. The figures give the optimal search paths which require returning to base during the search to recharge the battery. By increasing the capacity of the battery, the optimal strategy does not need to recharge. Figures 3(c) and 4(c) present optimal strategies for this

situation. In each case, the time to return to base when the object is found must be taken into consideration as opposed to only the time it takes to search.

Scenario 4: Control of Sensors. In this scenario we extend the power of the controller: as well as choosing the search path and when to recharge it can decide whether the search sensors are in a low or high power mode. In the high power mode the agent can search a cell, while in the low power mode it is only possible to traverse the cell. The high power mode for search is expensive in terms of time and battery use and can be unnecessary, e.g. when travelling over previously explored cells or returning to base to deposit or recharge. Again we assume the base and depot are fixed and located at position $(0, 0)$. The PRISM model for this scenario extends that for Scenario 3 as follows. A variable c is added to the agent module, taking value 0 and 1 when its sensors are in low and high power modes respectively. The (nondeterministic) choices of the controller are then extended such that when deciding the direction of movement it also decides the power mode of the sensors for traversing the next cell. To aid analysis of the synthesised strategies, the action labels for direction of search include the power mode of the sensors, e.g. *south1* corresponds to moving south and selecting high power mode and *west0* to moving west and selecting low power mode.



(a) path 1 and recharge (b) path 2 after recharge (c) path (no recharge)

Fig. 4. Scenario 3: optimal controllers for 5×4 grid, battery capacities 28 and 32.

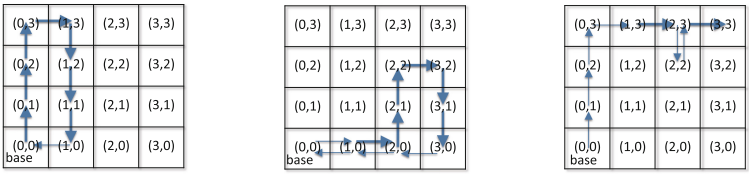
The PRISM code extract in Fig. 5 gives commands for moving east from cell with gridpoint 0 based on those in Fig. 2 for Scenario 3. The first two commands consider the case where the agents sensors are in high power mode ($c = 1$) and the cell is unexplored. In both cases, since the sensors are in high power mode and the cell is unexplored, the probability of finding an object is as for Scenario 3. The difference is that in the first command the sensors are switched to lower power mode, while in the second the sensors remain in high power mode. The third and fourth commands represent the case when the sensors are in lower power mode and the cell is unexplored. Since the sensors are in lower power mode, the cell remains unexplored and there is no chance of finding the object. The final two commands consider the case where the cell has been previously explored. The PRISM model is also updated so that the time passage and battery consumption reflects the sensor's current power mode.

```
// sensors in high power mode and cell unexplored
[east0] s=0 & c=1 & gp=0 & gp0=1 & posx<X &p≤1 →
   p : (s′=1)&(posx′=posx+1)&(gp0′=0)&(c′=0) + 1−p : (posx′=posx+1)&(gp0′=0)&(c′=0);
[east1] s=0 & c=1 & gp=0 & gp0=1 & posx<X &p≤1 →
   p : (s′=1)&(posx′=posx+1)&(gp0′=0)&(c′=1) + 1−p : (posx′=posx+1)&(gp0′=0)&(c′=1);
// sensors in lower power mode and cell unexplored
[east0] s=0 & c=0 & gp=0 & gp0=1 & posx<X → (posx′=posx+1)&(c′=0);
[east1] s=0 & c=0 & gp=0 & gp0=1 & posx<X → (posx′=posx+1)&(c′=1);
// cell already explored (does not matter the sensors power mode)
[east0] s=0 & gp=0 & gp0=0 & posx<X → (posx′=posx+1) & (c′=0);
[east1] s=0 & gp=0 & gp0=0 & posx<X → (posx′=posx+1) & (c′=1);
```

Fig. 5. PRISM commands: searching cell 0, moving east and switching sensors off/on.

Table 3. Scenario 4: model checking results.

Grid size	No. of objects	Battery capacity	States	Transitions	Min expected mission time	Verification time (s)
3 × 3	1	12	53,367	222,557	12.78	0.206
3 × 3	1	16	80,107	351,209	12.11	0.272
3 × 3	2	12	103,063	435,302	19.83	0.365
3 × 3	2	16	154,911	687,246	18.88	0.420
4 × 4	1	18	18,445,790	90,303,355	21.88	58.58
4 × 4	1	24	27,587,864	139,945,165	20.50	83.60
4 × 4	2	18	36,379,747	180,055,826	32.22	124.8
4 × 4	2	24	54,464,317	279,117,740	30.60	181.9



(a) first path (b) second path (c) third path

Fig. 6. Scenario 4: optimal controller for 4 × 4 grid, battery capacity 18 and one object.

Table 3 presents model checking results for this scenario including both those for the battery capacity from Scenario 3 (see Table 2) and for the minimum battery capacity required for a successful mission. Comparing with Table 2, allowing low and high power modes reduces the mission time, allows the mission to be completed with a smaller battery capacity and reduces recharging.

Comparing optimal strategies for Scenario 3 in Figs. 3 and 4 and those for Scenario 4 with the same battery capacity, the only difference is the low power mode is used when revisiting a cell. In Fig. 6 we present an optimal strategy for a 4×4 grid and battery capacity of 18. In this case, it is not feasible to complete the mission without using the lower power mode. Smaller arrows represent when the

sensors are in lower power mode. The move south during the third path before searching the final cell $(3, 3)$ might not appear optimal. However, immediately before this step there is an equal chance of finding the object in the two remaining unexplored cells $(2, 2)$ and $(3, 3)$. By moving south after searching $(2, 3)$ the time of returning to base is reduced when the object is found, at the cost of increasing the time to reach and search $(3, 3)$ when the object is not found. In fact it is the case that initially moving east from $(2, 2)$ also yield an optimal strategy, but was not the strategy synthesised by PRISM.

Scenario 5: Control of Multiple Agents. We now consider the case where there are multiple agents working together. We extend the PRISM model for Scenario 4 by having modules for two agents. In addition, since more than one cell can be explored at the same time, to simplify the PRISM code each cell is modelled as a separate module. The probability of finding an object is now dependent on both agents, and therefore we model this in a separate module.

```
// module for agent1
module agent1
   pos1x : [0..X] init basex; // x coordinate of agent1
   pos1y : [0..Y] init basey; // y coordinate of agent1
   // search (remain on grid and have sufficient battery)
   [search] u1=0 & u2=0 & pos1x<X & move_east → (pos1x'=pos1x+1); // east
   [search] u1=0 & u2=0 & pos1x>0 & move_west → (pos1x'=pos1x−1); // west
   [search] u1=0 & u2=0 & pos1y<Y & move_north → (pos1y'=pos1y+1); // north
   [search] u1=0 & u2=0 & pos1y>0 & move_south → (pos1y'=pos1y−1); // south
   // found object and not at base (go back to base)
   [end] (u1=1|u2=1) & !(agent1=base) → (pos1x' = basex) & (pos1y' = basey);
   // mission complete
   [end] (u1=1|u2=1) & agent1=base → true;
endmodule
// agent2 (rename agent1)
module agent2 = agent1[pos1x=pos2x, pos1y=pos2y, b1=b2] endmodule
```

Fig. 7. PRISM code: modules for *agent1* and *agent2* of Scenario 5.

The agent modules are presented in Fig. 7. Variables *pos1x* and *pos1y* represent the position of the first agent and *pos2x* and *pos2y* the second. Constants *basex* and *basey* give the position of the base and formula *base* the corresponding gridpoint. The search commands from the previous scenarios are modified and now synchronise on the action *search* with the gridpoint modules. Each command checks the variables *u1* and *u2* which indicate if an object has been found (see Fig. 7), since once the object is found, the agents return to base as the mission is complete. As the direction of movement is not encoded in the action *search*, preventing an agent moving in directions from which it cannot return to base with its remaining battery power is encoded in formulae *move_east*, *move_west*, *move_north* and *move_south* instead of the battery module.

For each cell in the grid there is a corresponding gridpoint module. The gridpoint modules for a 3×3 grid are presented in Fig. 8. By using the constants

```
// constants used for renaming gridpoints
const int k0 = 0;
        ⋮
const int k8 = 8;
// module for gridpoint 0
module gridpoint0
  gp0 : [0..1] init 1; // status of gridpoint0 (0 - explored and 1 - unexplored)
  // one of the agents searches the cell
  [search] (agent1=k0|agent2=k0) & (gp0=1) → (gp0'=0);
  // cell already searched or not being searched
  [search] !((agent1=k0|agent2=k0) & (gp0=1)) → true;
endmodule
// construct further gridpoints by renaming gridpoint0
module gridpoint1 = gridpoint0[gp0=gp1, k0=k1] endmodule
                  ⋮
module gridpoint8 = gridpoint0[gp0=gp8, k0=k8] endmodule
```

Fig. 8. PRISM code: module for gridpoints of Scenario 5.

```
// current gridpoint of agent1 and agent2 (derived from coordinates)
formula agent1 = pos1x+pos1y*(X+1);
formula agent2 = pos2x+pos2y*(X+1);
// retrieval probabilities module
module probabilities
  u1 : [0..1] init 0; // agent1 finds the object
  u2 : [0..1] init 0; // agent2 finds the object
  // agent1 and agent2 are searching different unexplored cells
  [search] s1=1 & s2=1 & agent1!=agent2 & 2*p≤1 → p : (u1'=1)
                                    + p : (u2'=1) + 1−2*p : true;
  // agent1 and agent2 searching the same unexplored cell
  // suppose each has the same chance of finding the object
  [search] s1=1 & s2=1 & agent1=agent2 & p≤1 → p/2 : (u1'=1)
                                    + p/2 : (u2'=1) + 1−p : true;
  // agent1 is searching unexplored cell while agent2 is not
  [search] s1=1 & s2=0 → p : (u1'=1) + 1−p : true;
  // agent2 is searching unexplored cell while agent1 is not
  [search] s1=0 & s2=1 → p : (u2'=1) + 1−p : true;
  // neither agent searching an unexplored cell
  [search] s1=0 & s2=0 → true;
endmodule
```

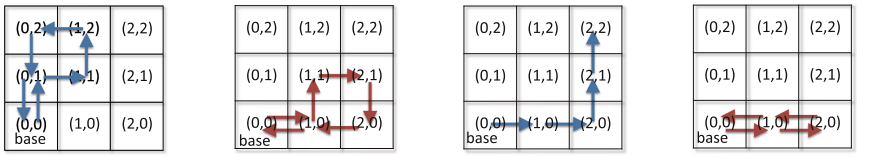
Fig. 9. PRISM code: retrieval probabilities module of Scenario 5.

ki we only need to explicitly construct the first gridpoint module and then use renaming. In the module for the first gridpoint (see Fig. 8), variable *gp0* is 1 when the cell is unexplored and 0 otherwise.

As stated above the probability of an agent finding an object is now a separate module, presented in Fig. 9. As before, the probability of an unexplored cell containing an object equals $1/n$ where n is the number of unexplored cells (see Fig. 1). Formulae *s1* and *s2* evaluate to 1 if *agent1* and *agent2* are searching unexplored cells respectively. If the agents are searching different unexplored cells, then each agent has a chance of finding the object, but both cannot find the object as the object cannot be in two places at once.

Table 4. Scenarios 5 and 6: model checking results.

	Grid size	Battery capacity	States	Transitions	Min expected mission time	Verification time (s)
Scenario 5	3×3	16	53,832	249,588	12.00	0.285
	4×4	24	15,555,103	91,859,618	17.50	52.53
	5×4	28	293,118,691	1,861,895,602	20.40	1,069
Scenario 6	3×3	16	153,063	910,392	11.93	0.701
	4×4	24	38,165,612	255,234,876	17.46	148.0
	5×4	28	691,136,157	4,826,058,012	20.36	14,671



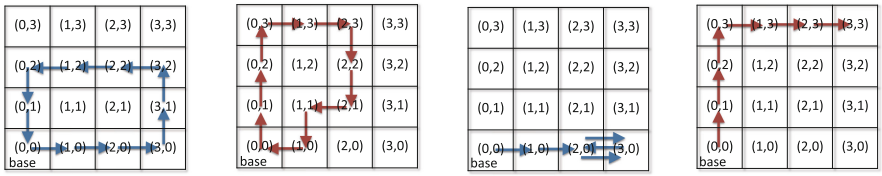
(a) path 1 (*agent1*) (b) path 1 (*agent2*) (c) path 2 (*agent1*) (d) path 2 (*agent2*)

Fig. 10. Scenario 5: optimal controller for 3×3 grid, battery capacity 16 and one object.

Table 4 presents model checking results for Scenario 5. As expected we see that searching with two agents can reduce the mission time over a single agent (see Table 2). Figures 10 and 11 present optimal strategies for grids of size 3×3 and 4×4. The optimal strategies are represented by the paths of the two agents before the object is found. As for the previous scenarios, as soon as the object is found the agents return directly to base. In both cases it is feasible for the agents to search the grid without recharging their batteries. However, this is not optimal due to the time required to return to base after finding the object. Neither the second path of *agent2* in Fig. 10 nor the second path of *agent1* in Fig. 11 contribute to the search. In both situations after recharging, there is only one cell to search ($(2,2)$ and $(3,3)$ respectively) and there is no gain in sending more than one of the agents to search this cell.

Scenario 6: Control of Multiple Agents with Idle Mode. As just discussed for Scenario 5, in certain situations there is no gain in both agents searching. For this reason in this scenario we add the ability for the controller to search using only one agent while the other idles at the base. Although this cannot reduce the mission time it can reduce power consumption and wear and tear.

Idling is introduced to the PRISM models through additional variables and the reward structure for time passage is updated to reduce the reward gained when an agent idles (see [37]). The optimal strategy will then choose idling over unnecessary movement, however as the reward is not reduced significantly it will use both agents to search when this can save time.



(a) path 1 (*agent1*) (b) path 1 (*agent2*) (c) path 2 (*agent1*) (d) path 2 (*agent2*)

Fig. 11. Scenario 5: optimal controller for 4×4 grid, battery capacity 24 and one object.

Table 4 includes model checking results for Scenario 6 (and 5). The reduced mission time from Scenario 5 to 6 is due to the change made to the reward structure and the generated optimal strategies yield the same expected mission time as those synthesised for Scenario 5. Figure 12 presents optimal strategies for a grid of size 3×3. This strategy is very different from that for Scenario 5 as in this case *agent1* searches the majority of the grid, while *agent2* searches only a small portion and returns to base and idles while *agent1* completes its search. For the 4×4 grid the optimal strategy is initially the same as for Scenario 5 (see Fig. 11). However, in the second phase of the search there is no path for *agent1*, instead it idles at base while *agent2* searches the remaining cell.

(a) path (*agent1*) (b) path (*agent2*)

Fig. 12. Scenario 6: optimal strategy for 3×3 grid, battery capacity 16 and one object.

4 Conclusions and Future Work

We have demonstrated that probabilistic model checking and PRISM can be used for the synthesis of controllers for autonomous agents. However, there are clearly scalability issues as the models generated can have hundreds of millions of states for simple scenarios. Therefore, to analyse real-world applications, abstraction (and refinement) techniques are required. In particular, we will investigate using the game-based abstraction approach of [16] in this context, as well as symmetry reduction techniques [25] as there is symmetry both in the environment, e.g. in a grid structure, and between agents. Regarding the formal models and specifications, improving the efficiency of the POMDP implementation [26] could have significant modelling benefits, as in real applications control decisions must

be based only on the information from sensors, and therefore only on a partial view of the environment. Stochastic games are also required to model and separate the nondeterminism present in the environment from the choices of the controller. Combining these aspects will require the analysis of partially observable stochastic games which are harder to solve than POMDPs [3]. PRISM has support for multi-objective queries [9] and this will allow the synthesis of more specific controllers, e.g. that optimise the mission time while limiting both power consumption and failure, and ensuring safety requirements.

As for using PRISM for the analysis, the current way optimal strategies are exported can be improved. In particular, having a graphical representation would have simplified the analysis. In addition, allowing the analysis of a synthesised strategy directly would have saved considerable effort. Currently, to do this, the strategy has to be exported to a file and then imported back into PRISM (together with the state space and reward structures).

The PRISM models were developed from Simulink models [14]. PRISM generates the synthesised controllers as a lists of reachable states and optimal action choices for the states, this output from PRISM can be easily fed back into the *search* module of the Simulink models. Currently we are adapting the models for a larger search area consisting of adjoined discrete, symmetric regions. The same controller can then be used (modulo a symmetry transformation) on each region in turn until all objects have been located. At present the creation of UAV software from the Simulink code is automatic, future work will involve the direct embedding of the PRISM generated controllers into the UAV software.

Acknowledgements. This work was supported by EPSRC grant EC/P51133X/1. We would like to thank Dave Anderson and Euan McGookin for discussions on the autonomous systems that inspired this paper.

References

1. Alur, R., Henzinger, T.: Reactive modules. FMSD **15**, 7–48 (1999)
2. Bohn, C.: Heuristics for designing the control of a UAV fleet with model checking. In: Grundel, D., Murphey, R., Pardalos, P., Prokopyev, O. (eds.) Cooperative Systems. Lecture Notes in Economics and Mathematical Systems, vol. 588, pp. 21–36. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-48271-0_2
3. Chatterjee, K., Doyen, L.: Partial-observation stochastic games: how to win when belief fails. ACM Trans. Comput. Log. **15**, 16 (2014)
4. Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., Simaitis, A.: PRISM-games: a model checker for stochastic multi-player games. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 185–191. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36742-7_13
5. Choi, J.: Model checking for decision making behaviour of heterogeneous multi-agent autonomous system. Ph.D. thesis, Cranfield University (2012)
6. Dennis, L., Fisher, M., Lincoln, N., Lisitsa, A., Veres, S.: Practical verification of decision-making in agent-based autonomous systems. ASE **23**(3), 305–359 (2016)
7. Ding, X., Smith, S., Belta, C., Rus, D.: Optimal control of Markov decision processes with linear temporal logic constraints. IEEE Trans. Autom. Control **59**, 1244–1257 (2014)

8. Draeger, K., Forejt, V., Kwiatkowska, M., Parker, D., Ujma, M.: Permissive controller synthesis for probabilistic systems. LMCS **11**(2), 1–34 (2015)
9. Etessami, K., Kwiatkowska, M., Vardi, M., Yannakakis, M.: Multi-objective model checking of Markov decision processes. LMCS **4**, 1–21 (2008)
10. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D.: Automated verification techniques for probabilistic systems. In: Bernardo, M., Issarny, V. (eds.) SFM 2011. LNCS, vol. 6659, pp. 53–113. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21455-4_3
11. Fu, J., Topcu, U.: Computational methods for stochastic control with metric interval temporal logic specifications. In: Proceedings of CDC 2015 (2015)
12. Hoffmann, R., Ireland, M., Miller, A., Norman, G., Veres, S.: Autonomous agent behaviour modelled in PRISM – a case study. In: Bošnački, D., Wijs, A. (eds.) SPIN 2016. LNCS, vol. 9641, pp. 104–110. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-32582-8_7
13. Humphrey, L.: Model checking for verification in UAV cooperative control applications. In: Fahroo, F., Wang, L., Yin, G. (eds.) Recent Advances in Research on Unmanned Aerial Vehicles. LNCIS, vol. 444, pp. 69–117. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37694-8_4
14. Ireland, M., Hoffmann, R., Miller, A., Norman, G., Veres, S.: A continuous-time model of an autonomous aerial vehicle to inform and validate formal verification methods. http://arxiv.org/abs/1609.00177v1
15. Kalra, N., Paddock, S.: Driving to safety: how many miles of driving would it take to demonstrate autonomous vehicle reliability? Transp. Res. Part A: Policy Pract. **94**, 182–193 (2016)
16. Kattenbelt, M., Kwiatkowska, M., Norman, G., Parker, D.: A game-based abstraction-refinement framework for Markov decision processes. FMSD **36**, 246–280 (2010)
17. Kemeny, J., Snell, J., Knapp, A.: Denumerable Markov Chains. Springer, New York (1976). https://doi.org/10.1007/978-1-4684-9455-6
18. Konur, S., Dixon, C., Fisher, M.: Analysing robot swarm behaviour via probabilistic model checking. Robot. Auton. Syst. **60**(2), 199–213 (2012)
19. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47
20. Kwiatkowska, M., Parker, D.: Automated verification and strategy synthesis for probabilistic systems. In: Van Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 5–22. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-02444-8_2
21. Lacerda, B., Parker, D., Hawes, N.: Multi-objective policy generation for mobile robots under probabilistic time-bounded guarantees. In: Proceedings of ICAPS 2017 (2017)
22. Lahijanian, M., Andersson, S., Belta, C.: Formal verification and synthesis for discrete-time stochastic systems. IEEE Trans. Autom. Control **60**, 2031–2045 (2015)
23. Lahijanian, M., Kwiatkowska, M.: Specification revision for Markov decision processes with optimal trade-off. In: Proceedings of CDC 2016. IEEE (2016)
24. Liu, W., Winfield, A., Sa, J.: Modelling swarm robotic systems: a case study in collective foraging. In: Proceedings of TAROS 2007 (2007)
25. Miller, A., Donaldson, A., Calder, M.: Symmetry in temporal logic model checking. Comput. Surve. **36**, 8 (2006)

26. Norman, G., Parker, D., Zou, X.: Verification and control of partially observable probabilistic systems. Real-Time Syst. **53**, 354–402 (2017)
27. O'Brien, M., Arkin, R.C., Harrington, D., Lyons, D., Jiang, S.: Automatic verification of autonomous robot missions. In: Brugali, D., Broenink, J.F., Kroeger, T., MacDonald, B.A. (eds.) SIMPAR 2014. LNCS (LNAI), vol. 8810, pp. 462–473. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11900-7_39
28. Shapley, L.: Stochastic games. Proc. Natl. Acad. Sci. **39**, 1095–1100 (1953)
29. Sharan, R.: Formal methods for control synthesis in partially observed environments: application to autonomous robotic manipulation. Ph.D. thesis, California Institute of Technology (2014)
30. Soudjani, S., Majumdar, R.: Controller synthesis for reward collecting Markov processes in continuous space. In: Proceedings of HSCC 2017. ACM (2017)
31. Svoreňová, M., Chmelík, M., Leahy, K., Eniser, H., Chatterjee, K., Černá, I., Belta, C.: Temporal logic motion planning using POMDPs with parity objectives: case study paper. In: Proceedings of HSCC 2015. ACM (2015)
32. Svoreňová, M., Křetínský, J., Chmelík, M., Chatterjee, K., Cerna, I., Belta, C.: Temporal logic control for stochastic linear systems using abstraction refinement of probabilistic games. In: Proceedings of HSCC 2015. ACM (2015)
33. Webster, M., Fisher, M., Cameron, N., Jump, M.: Formal methods for the certification of autonomous unmanned aircraft systems. In: Flammini, F., Bologna, S., Vittorini, V. (eds.) SAFECOMP 2011. LNCS, vol. 6894, pp. 228–242. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24270-0_17
34. Wilson, J.: Drones hacked and crashed by research team to expose design flaws. Engineering and Technology (2016)
35. Wolff, E., Topcu, U., Murray, R.: Robust control of uncertain Markov decision processes with temporal logic specifications. In: Proceedings of CSC 2012. IEEE (2012)
36. Yoo, C., Finch, R., Sukkarieh, S.: Provably-correct stochastic motion planning with safety constraints. In: Proceedings of ICRA 2013. IEEE (2013)
37. http://www.prismmodelchecker.org/files/nfm18/